Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 85 (2016) 352 – 361

Procedia Computer Science

International Conference on Computational Modeling and Security (CMS 2016)

An UML+Z framework for validating and verifying the Static aspect of Safety Critical System

Monika Singh[a], Ashok Kumar Sharma[a], Ruhi Saxena[b]

[a]Faculty of Engineering & Technology (FET) Mody University of Science and Technology, India

[b]Computer Science & Engineering (CSE), Thapar University, India

Abstract

The aim of this paper is to propose an augmented framework for verifying and validating the static aspect of safety critical systems by analysing the UML class diagrams and the relationship between them. Since UML is a semi formal language which is provn to ambiguities due to its various graphical notations, hence Formal analysis of UML class diagram is required. Moreover, class diagram play an important role in system designing phase especially in safety critical systems. Any ambiguity or inconsistency in design can result in potential failure. Formal methods are the mathematical tools and methodology which are sandwiched at various stages of software development process to ensure the correctness, consistency and completeness of software artifacts such as requirement specifications, design etc. In this article, Z notation is used for the purpose of analysis formally and later on verified by the Z/EVES tool.

© 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of CMS 2016

Keywords: UML class diagram; Class; Relation; Formal method; Z Notation; Z/EVES.

1. Introduction

Nevertheless, in software development process, all the phases have their own importance, but the requirement and implementation phase are most critical one. Unified modelling language which is a semi formal modelling language, used to analyse requirements and visualise the design of software artefacts. Although UML is used as a de facto standard for designing the software, but due to semiformal in nature, chances are that ambiguities might be introduced. Some of the issues with UML are mentioned below [2]:

1) UML structures are based on graphical notations and are prone to causing errors,
2) The hidden semantics of UML allows ambiguities at design level,
3) The same system needed can be described by multiple notations or diagrams which may cause inconsistencies or ambiguities
4) UML model may have multiple interpretations

One way of resolving these issues is to use a formal model in integration with UML. Formal methods use the discrete mathematics which includes set theory, first order predicates, logics and graphs.

* Corresponding Author. Tel.: +91-9549446789.
E-mail address:Dhariwal.monika@gmail.com

1877-0509 © 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of CMS 2016
doi:10.1016/j.procs.2016.05.243

The lack of preciousness in UML semantics can be covered by the rigorous mathematics used by the formal methods by integrating UML with formal model, the expressiveness of graphical notation increases which ultimate enhance the modeling power of UML diagrams especially at analysis and designing part. The UML class diagrams play an important role at designing part of system. By making formal model of UML class diagrams, inconsistency and ambiguities can be reduced to a large extent which will ultimately reduced the cost of re implementation due to incompleteness and ambiguities in system design specifications. However due to having rigorous mathematics use, formal methods are not warm welcomed by industry's peoples, although formal methods are highly recommended for safety critical real time application [3] [7]. Moreover, formal methods are equipped with tool, which can be used for both the prospective i.e. for describing a system and later on for analyzing their functionalities. Although, formal methods do not guarantee correctness, but their use emphasize to increase understanding of a system by divulging errors or facets of incompleteness that might be expensive to correct them at any later point of time.

Various formal languages are used for this purpose like VDM, B-Methods, Petri Net, and Z Notation etc.

Z notation is a model based formal specification language which uses the set theory and first order predicates and having a strong tool support. In this paper, the relationships of the UML class diagrams are recognized and mapped to Z schema. The class diagram is composed of classes and their relations with other classes in the system.

The organization of the rest paper is as follow: In Section 2, Literature review has been discussed. Methodology and tool used are briefly described in section 3. In section 4, the proposed work i.e. integration of Class Diagram and Z notation is given. Result analysis is done by formal model in section 5. In last, the section 6, wind up with conclusion and future scope.

2. Literature Review

A lot of work exists in the field of the integration of UML with formal methods [4-9] [10] [31]. In [4], a framework is proposed which explain how by using Z notation in inventory system, the design and development can improve the quality effectively and reduce cost. A comparative case study is done for formal languages i.e. Z Notation, UML, SDL, LOTOS and UML and compared their key characteristics by designing a particular system for each formal language in article [5].in paper [6], a tool is developed which takes UML class diagram in the form of petal files, ASCII format files generated by Rational Rose, and evaluates it automatically and produces a list of comments. In an article [7], the safety critical systems are modelled by combine two formal language i .e. Z notation and Perti net. The paper [8] advocate a Formal Approach to Validating and Verifying Functional Design for Unmanned Aerial System Remote Sense, Department of Defence Federal Initiative, Joint Unmanned Aircraft Systems Center of Excellence at Creech Air Force Base, Nevada, DoD Contract Number FA4861-07-R-C003 which is a complex safety critical system. [15] Gives an overview of tool RoZ in which the transition between the UML world and the Z world: from an annotated class diagram, it automatically generates a complete Z specification. In paper [20], a comparative study has been done for various formal languages based upon various parameters. Some more related work mentioned in [9] [16] [17] [18] [19] [10-14].

3. Methodology and Tool

The proposed framework is shown in figure 1:

The methodologies used in this paper are UML and Z notation which are disused as follow:

3.1 UML: Unified Modeling Language [1] is the de-facto standard for analysis, constructing and visualizing the artifacts in object oriented paradigm. UML diagrams are broadly divided into two categories which further consist of fourteen diagrams as:

a. Structural Diagrams: Composite structure diagram, Deployment Diagram, Package diagram, Profile diagram, Class diagram, Object diagram, Component diagram

b. Behavioral Diagram: State machine diagram, Communication diagram, Use case diagram, Activity diagram, Sequence diagram, Timing diagram, Interaction diagram.

The UML class diagram comes under the category, Structural Diagrams which in turn represents the static view of system. As Class diagram represents the object orientation view of a system therefore it is widely used at construction. Any ambiguities at design level can result into hazard. In this paper, a formal model for class diagram is proposed to avoid such ambiguities.

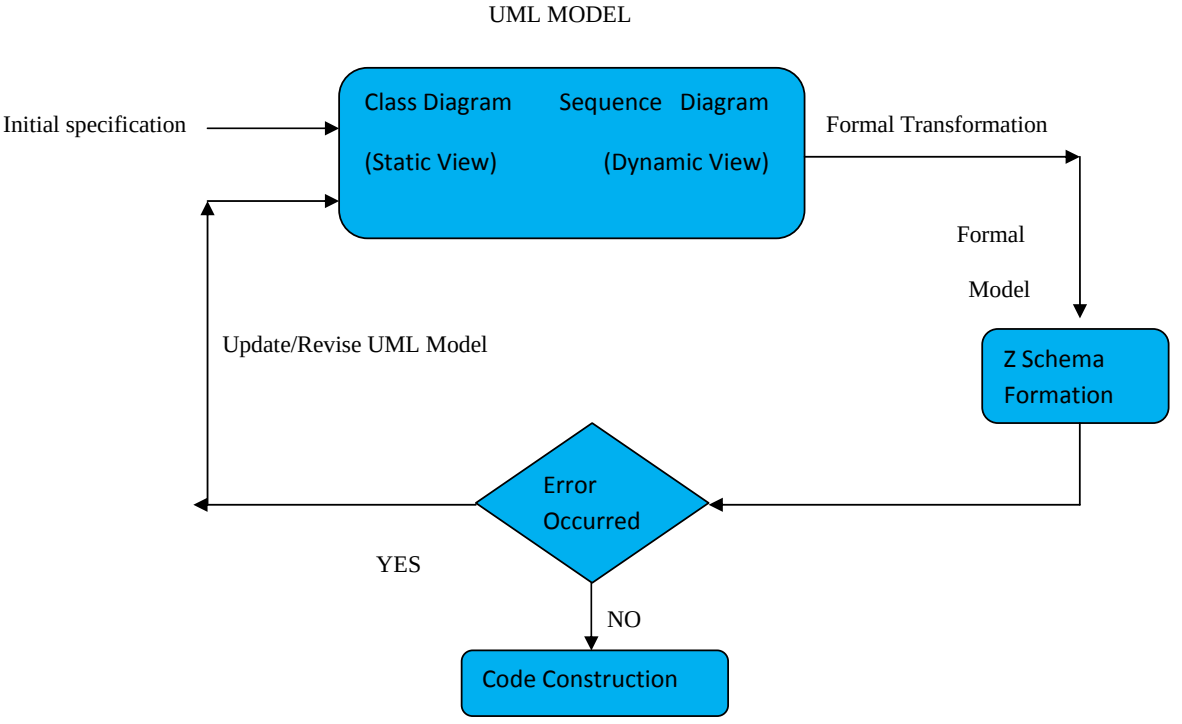

Fig 1: UML+Z Framwork for validating the static aspect

3.2 Formal method

Formal methods [24-28] are methodologies based upon mathematical techniques and equipped with tool for enhancing the quality of system or software by incorporating accuracy, completeness, and consistency in specifications. In other words, formal methods provide a structure which help engineers to specify, develop and verify the system in a systematic way. Formal methods are grouped into two types:

Model-oriented methods: In model oriented methods , the system's specification are defined as a model by using mathematical notion such as set , relation, sequences and functions. VDM, B, and Z notation, Petri net, Communicating Sequential processes (CSP), Calculus of Communicating Systems (CCS), and I/O automata.

Property-oriented methods: On the other hand, the property oriented methods make use of axioms to satisfy the system properties. OBJ, LOTOS is formal languages that belong to this category.
The Z notation [29-30] is used for the purpose of formal analysis of class diagrams in this paper.
The use of formal methods in system development is derived by the fact that proper analysis by mathematical theorems leads to the correctness, consistency and completeness in proposed system's specifications. Moreover, the relationship between class diagram and Z notation is found good as classes in class diagram can be described in terms of relationship between schemas of Z. The generic schema structure is shown in figure 2:

Schema Name

Variables Declared

Invariant
(*relationship between the values of variables*

Fig. 2: Z Notation Schema Structure

The Z specification is further verified with the tool Z/EVES. By using Z/EVES, following can be done: 1) syntax and type checking, 2) schema expansion, 3) precondition calculation, 4) domain checking, and 5) general theorem prove.

4. Formal Model of Class Diagram

The class diagram [21] is a collection of classes and the existing relationship between these classes. A class must have attributes and type of attribute can be public, private or protected. Therefore the set of all classes, type and relation are considered as basic type of specification. .

[N, Type, Rel]

Class

name : *N*
type: *Type*
rel: *Rel*

Type::= base class subclass
Attribute Type::= public | private | protected

The collection of classes is represented by the schema Class *Diagram* which composes of three variables i.e. base class, sub class and relat. The mapping *relat* from *base class to* power set of *Class* describes types of the class.

ClassDiagram

Base class: *Class*
Subclass: *S Class*
relat: *Class* ▪ *S Class*

base class ≍ *class* ◊ *class* ⊥ ! *subclasses*
base class ⌔ *dom subclasses* ◊ *subclasses* ⊥ ≅
≅ *c*: *Class c* ≍ *dom subclasses* ▪ ! *c* ≍ *classes* ≅ *c*: *Class c* ≍ *classes*
▪ *c* ⊥ *base class* ◊ *c* ⊥ *subclass* ∫∫ ! *c* ≍ dom *subclasses* ◊ *base class* ⌔ ! dom *subclasses*

Since the below part of central line of schema are invariants, therefore the invariants in class diagrams are:

(i) The base class is not in the power set of classes (ii) The base class and the subclass should not be identical (iii) The base class should not belong to the domain of subclasses to prove that it is not a subclass (iv) The power set of classes is non-empty (v) for any class s. t it lies in domain of subclasses, it belongs to the collection of classes.

To move from one class to another, relation must be established between the base class and the subclasses. The relation consists of three components i.e. type, cardinality and location. Location is the values of those two classes between whom the relation exists. The relation is defined by a schema *Relationship* in Z which consists of three variables that are *type, cardinality and location* as given blow:

Relation Name::=association | aggregation | generalization | dependency

Cardinality::= (0 ....1) | (1.....1) | (1*.....n)

Location::= (Base class to subclass) | (subclass to subclass) | (base class to base class) | (subclass to sub class)

Relation

relname : *RN*
cardinality: *Cardinality*
location: *Location*

The Relationship between the different classes is represented by the schema *Relationship*. The relationship function takes a class and relation and returns the same or a new class.

The set of different type of relation is represented by the schema *RelName* which further consist of *ClassDiagram* and *Relationship schemas.* . The *ClassDiagram* presents the set of classes and *Relationship* presents all possible relations among the classes. The invariants of schema *Relationship* are following:

(i) For every relation in the set all possible relationship, there exits two classes and relation between these classes s. t. The relname is in the set of relation (ii) for any two classes and a relation over the classes, there exists relation name (relname), cardinality and location s. t. the relation is in the domain of relationship and must be mapped from one class to another after once the relation is defined.

Relationship

relname: *RN*
relationship: *Class* ↛ *Relation* ▪ *Class*

≅ *re*: *RN re* ≍ *relname*
▪ ≈ *c*1, *c*2 : *Class* ; *rel* : *Relation*, (*c*1, *rel*) ≍ dom *relationship*
◊ *c*2 ≍ ran *relationship* ◊ *relationship* (*c*1, *rel*) = *c*2 ▪ *re* = *rel* ▪ *relanme* ≭ *re* ≍ ran *rel* ▪ *location*
≅ *re*: *RN re* ≍ !*relname* ▪ ≈ *c*3, *c*4: *Class*; *rel*: *Relation*
(*c*3, *rel*) ≍ dom *relationship* ◊ *c*4 ≍ !ran *relationship* ◊ *relationship* (*c*3, *rel*) = *c*4
▪ *re* = *rel* ▪ *relanme* ≭ *re* ≍ ran *rel* ▪ *location*

The set of relation type of the class diagram is represented by the schema *RelName* which consists of *ClassDiagram* and *Relationship* schemas. The Class*Diagram* corresponds to set of classes and *Relationship* corresponds to all the possible relations among the classes defined in class diagram.

RelName

ClassDiagram
Relationship

≅ *c*1 : *Class c*1 ≍ *classes* ≍ *classes* ◊ *c*1 ⊥ *Baseclass*
▪ ≈ *c*2 : *Class*; *rel*: *Relation* (*c*2, *rel*) ≍ *dom relationship* ▪ *c*1 = *c*2
≭ *c*3 : *Class*; ; *rel*: *Relation* (*c*3, *rel*) ≍ *dom relationship* ▪ ≈ *c*4 : *Class c*4 ≍ *classes* ▪ *c*3 = *c*4
≅ *c*5 : *Class*; *rel* : *Relation* (*c*5, *rel*) dom *relationship* ◊ *c*5 ▪ *type* ⊥ *Baseclass*
▪ ≈ *c*6: *Class c*6 ≍ *classes* ▪ *relationship* (*c*5, *rel*) = *c*6

In the Relationship diagram, it is possible that when a relation is exits, it results in to the same class for example, the Association Relation. The Equivalence property for Association relation is shown here in this

section one by one. A relation is said to have an equivalence property if and only if it has 1) Reflexive, 2) Symmetric and 3) Transitive property. Therefore, there exists a set of classes over which the equivalence relation is satisfied. The reflexive relation over the set of classes is defined in terms of the schema *ReflexiciveRelations*. It takes the *Relation* schema *as input* and verifies the reflexive property over the association relation. The invariants in the schema *Reflexcive Relation* are as follow:

(i) For every subclass, there is a relation which can be mapped over it. (ii) For every class with a relation, must be in the domain relationship. (iii) For every subclass there is a relation who acts on this subclass and results the same or a new subclass.

Reflexcive Relation

ώ *Relation*

refeRel: *Class* ↿ *Class*

≅ *c*: *Class*; *rels*: *Location c* ≍ *classes* ◊ ran *rels* ⋋ *relation*

▪ (*location*, *location*) ≍ *refeRel* ∮ (≈ *rel*: *Relation* (*c*, *rel*) ≍ dom *relationship*

▪ *relationship* (*c*, *rel*) = *c*◊ 1 ⊩ # *location* ◊ *rel*. *relname* = *location* 1 ◊ (≅ *x* : *N*, *x*≍ 1…..# *location* -1

◊ 1+ 1⊩ # *location* ◊ (*rel*, *location*, *x*) ≍ *appliesto* ▪ *location* (*x*+1) = *rel* ▪ *class x*))

A Symmetric relation R is defined over a set X if for all x, y in the set *X* if (x, y) is in the relation *R* then (y, x) is also in *R*. In class diagrams, it is possible that when a base class to subclass or any subclass to subclass, the order in which the classes are connected and control progress from one class C1 to other C2 subsequently symmetry forces and there exists an inverse of the order of connected classes which results C2 to C1. The schema *Symmetric Relation* defines the symmetric relation over the set of relation of class diagram which captures the schema *Relation* and validates the symmetric property.

The invariants for symmetric relation are as follow:

(i) For any two classes *c*1 and *c*2, and a sequence of location which move the state *c*1 to *c*2 after mapping in class diagram, for another class *c*3, and a sequence of locations which move the class c2 to class *c*3, there exists another sequence of relation which is amended form of the above sequences and it proceeds the class c1 to *c*3.

Symmetric Relation

ώ *Relation*

symRel: *Class* ↿ ǃ *Class*

≅ *c*1, *c*2: *Class*; *loc*1, *loc*2 : *Location c*1 ≍ *classes* ◊ *c*2 ≍ *classes* ◊ ran *loc*1 ⋋ *relation* ◊

*ran loc*2 ⋋ *relation* ▪ (*loc*1, *loc*2) ≍ *symRel*

∮ (≈ *rel*1, *rel*2: *Relation* (*c*1, *rel*1) ≍ dom *relationship* ◊ (*c*2, *rel*2) ≍ dom *relationship* ▪ *relationship*

(*c*1, *loc*1) = *c*2 ◊ *relationship* (*c*2, *rel*2) – *c*1◊ 1⊩ # *loc*1◊ 1⊩ #*loc*2 ◊ *rel*1 ▪ *location* = *loc*1

◊ (≅ *k*:*N*, *k*≍ 1….#*loc*1-1 ▪ ((*rel*1. *Location*, *k*)≍ *appliesto* ◊ ǃ*loc*1 (*k*+1) = *rel*1. *Location* = *loc*2 ◊

(≅ *k*: *N*, *k*≍ 1….# *loc*1-1 ▪ ((*rel*2.*location*,*k*) ≍ *appliesto* ◊ *loc*2(*k*+1) = *rel*2.*locationk*)))

A relation *R* over *X is said to be* transitive if for any i, j, k in *X*, and (i, j) in R and (j, k) in *R* the order pair (i, k) in the relation *R*. To show transitivity in class diagrams, if a relation is mapped from one class C1 to another class C2 and then a new relation is mapped from *C*2 to *C*3 then an amalgamated relation can be mapped from C1 to C3 which implies that the transitive relation exits over the association relations in class diagram. The formal definition of transitivity over the set of relations of the class diagram is defined by using the schema *Transitive Relation* which includes the schema *Relation* for verification of transitivity.

Transitive Relation

ώ *Relation*

transRel: *Class* ↓ *Class*

≅ *c*1, *c*3: *Class*; *loc*1, *loc*2: *Location*, *c*1≍ *classes* ◊ *c*3≍ *classes* ◊ 1 ⊩# *loc*1 ◊ 1⊩ #*loc*2

◊ ran *loc*1≻ *location* ◊ ran *loc*2≻ *location* ▪ (*loc*1, *loc*2) ≍ *transRel*

∯ (≈ *c*2:*Class*; *rel*1, *rel*2: *Relation*, *c*2 ≍ !*classes* ◊ !(*c*1, *rel*1) ≍ dom *relationship* ◊ (*c*2, *rel*2) ≍ dom *relationship*

▪ *relationship* (*c*1, *rel*1) = *c*2◊ *relationship* (*c*2, *rel*2) = *c*3 ◊ *rel*1. *Relname*= *loc*1 ◊ (≅ *k*:*N*, *k*≍ 1….#*loc*1-1

▪ ((*rel*1.*location*,*k*) ≍ *appliesto* ◊ *loc*1(*k*+1) = *rel*. *location k*))

◊ *rel*2. *Relname*= *loc*2 ◊ (≅ *k*:*N*, *k*≍ 1….#*loc*2-1 ▪ ((*rel*2.*location*,*k*) ≍ *appliesto* ◊ *loc*2(*k*+1) = *rel*. *location k*))

The list of invariants is as follow:

(i) For any two classes *c*1 and *c*2, and a sequence of relations which maps the class *c*1 to c2 in the class diagram, for other state *c*3, and a sequence of relations which maps the class c2 to class *c*3, there exists one more sequence of relations which is aggregation of the above sequences and their maps from class *c*1 to *c3.*

5. Case Study & Result Analysis

The Z specification of class diagrams are written by using the schema notion [30]. Further, the specifications are verified for syntax and consistency by the computer tool Z/EVEV [22]. Z/EVES present two type of interface: Graphical user interface and the Command line interface [22] [29-30]. In this paper, we used the Graphical user interface for verifying and composing the specification which were written in Z notation language. Moreover, Z/EVES propose two mode of operations i.e. “Eager” and “Lazy”. In our article we use the “Eager” mode since in this mode a paragraph is checked if and only if all the previous ones are checked which is highly recommended for safety critical real time application. The snapshots of results of formal analysis of class diagram are shown in figure 3.

The left most first column is for syntax checking and the second left most columns is for correctly implementing the proof. If the values in both the column is Y, that mean the given z specification having correct syntax and the proof is also correctly implemented.

Moreover, domain checking is also provided by the Z/EVES tool. In context to domain checking, Z/EVES allow one to write only meaningful statements. In Z/EVES the action point for proof provided by theorem prover to perform the domain checking may be from one of the following: reduce, normal, quantifier, case and equality. The action point in this paper is *“prove by reduce”*.

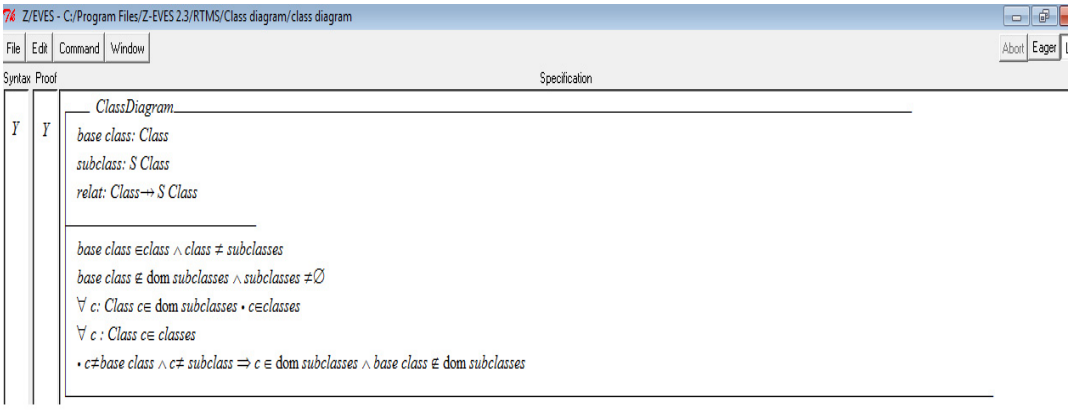

Fig 3: The snapshots of formal analysis of Class diagrams with Z/EVES tool

To implement this formal model however, Road traffic management system (RTMS) is taken as a case study in this paper. The UML class diagram of Road traffic management system is shown in figure 4:

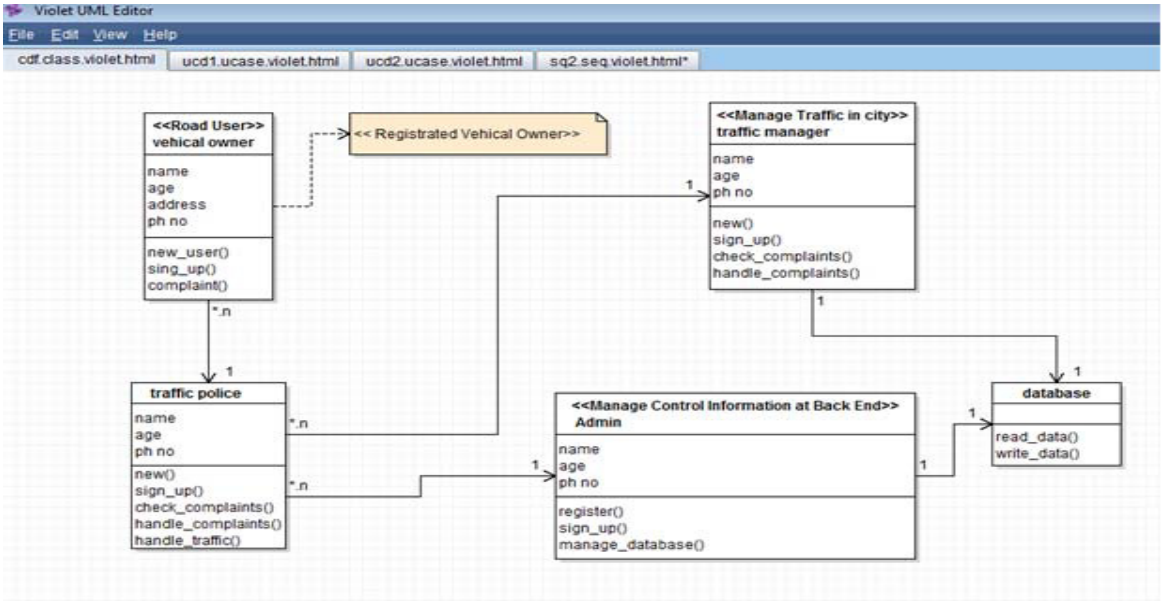

Fig 4: UML Class Diagram of road traffic management system

The main classes are: Traffic Police, Vehicle Owner Admin. The Vehicle owner class is considered for analysis here. The schema consists of two variables:

- *Vowner* is the set of names with RTMS registered and
- *registVowner* is the function which when implemented on a particular Vehicle Owner name, provides the unique registration number associated with the person.

In fig.5, the schema for Vehicle Owner with basic data type is given:

[Name, Seqchar]

VehicleOwner

Vowner: ix*Name*

Age: ix v

Phno: *Seqchar*

registVowner: *Name* ▪ !*Seqchar*

Vowner = dom *registVowner*

Fig 5: State Space of Vehicle Owner schema

In VehicleOwner schema, we define a partial function named “registVowner” which maps the corresponding vehicle owner with a registration number i.e.

registVowner: Name ▪ Seqchar

Moreover, **“registVowner”** is a one-to-one function which maps vehicle Owner name with registration number. Since it is a One-to-one function, therefore every Vehicle Owner has a unique registration number and consequently, would be no ambiguity.

The schema of Vehicle Owner is further verified by Z/EVES tool in fig. 5. The left most columns' value "Y" shows that the schema is implemented using correct syntax. If there would be any syntax error, it shows "N" instead of "Y" in syntax column [22]. Z/EVES also provide the schema expansion facility which help to understand the abstract semantics of schema which is shown in figure 6:

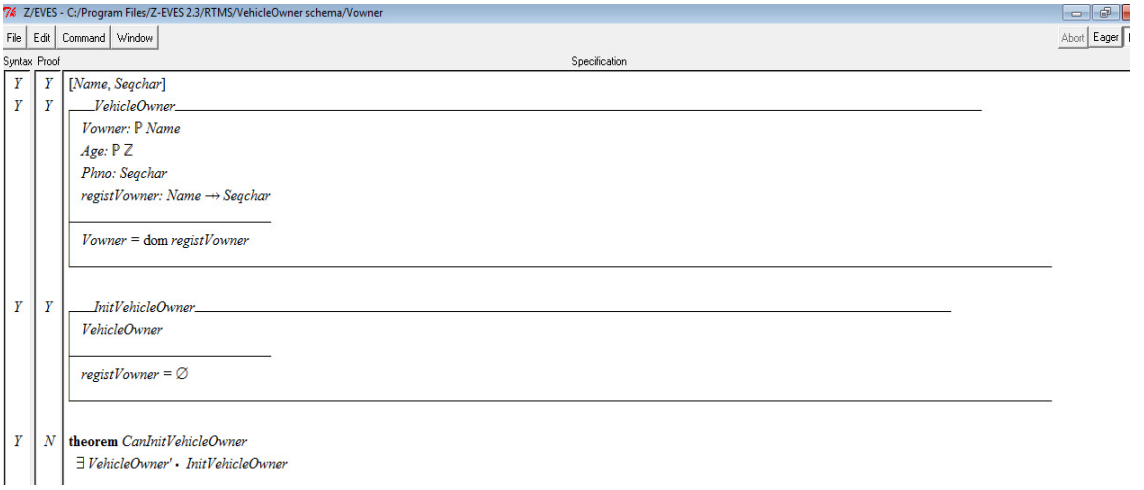

Fig 6: Snapshot of Formal Analysis of Vehicle owner with Z/EVES tool

The result of analysis for syntax and domain checking is shown in Table 1 as follow:

Table1. Result of analysis by Z/EVES Toolset

Schema Name	Syntax & Type checking	Domain Checking	Proof	Reduction
Class	Y	Y	Y	Y
Class Diagram	Y	Y	Y	Y#
Relation	Y	Y	Y	Y
Relationship	Y	Y	N	Y#
Reflexcive Relation	Y	Y	Y	Y
Symmetric Relation	Y	Y	Y	Y
Transitive Relation	Y	Y	Y	Y

The "#" symbol is used to show that the proof is done by reduction action point one of the facility available in Z/EVES toolset

6. Conclusions & Future work

The Behavioural specification, i.e. describing how things change is the core characteristic of Z specification but at the same times the loss part of UML. In UML, we can articulate state changes, collaboration, and workflow, but we cannot illustrate how objects change in terms of transformations. The positive part of UML is simplicity and the negative side of Z is a lack of graphical notation. Therefore, combining the two approaches is required for a better understanding of proposed system to cover ambiguity, identifying consistency and enhancing completeness. The UML class diagrams having their importance in designing phase of system development. The formalization of UML class diagrams for syntax and hidden semantics is indeed. Further, the formal model is verified by Z/EVES toolset which in turn provide the facilities of schema expansion, domain checking, syntax and type checking. In future, the formal model of sequence diagram will be proposed in order to capture the dynamic aspect of system.

Acknowledgments

Authors are thankful to Faculty of Engineering and Technology, Mody University of Science and Technology for providing the facility to carry out the research work and the numerous reviewers for their valuable suggestions.

References

[1].Rumbaugh, I. Jacobson and G. Booch, 2006 *The Unified* Modeling *Language Reference Manual, Second Edition*,
[2].N. H. Ali, Z. Shukur and S. Idris 2007. A Design of an Assessment System for UML Class Diagram, *Inter- national Conference on Computational Science and Applications*, Kuala Lumpur, (26-29 August) pp. 539- 546.
[3]. R. Miles and K. Hamilton 2006. *Learning UML 2.0* 1st Edition, O'Reilly Media, Sebastopol, pp 288.
[4]. R. Borges and A. Mota 2003. Integrating UML and Formal Methods, *Electronic Notes in Theoretical Computer Science*, Vol. 84, pp. 97-112
[5]. M. Brendan and J. S. Dong 1998. Blending Object-Z and Timed CSP: An Introduction to TCOZ, *Proceedings of International Conference on Software Engineering*, Kyoto, (19-25 April), and pp. 95-104.
[6]. W. S. Changchien, J. J. Shen and T. Y. Lin 2002. A Preliminary Correctness Evaluation Model of Object-Oriented Software Based on UML, *Journal of Applied Sciences*, Vol. 2, No. 3, pp. 356-365. doi:10.3923/jas.2002.356.365
[7]. A. Dennis, B. H. Wixom and D. Tegarden 2005. Systems Analysis and Design with UML, 3rd Edition, Wiley, New York, 576 pp.
[8]. Z. Derakhshandeh, B. T. Ladani and N. Nematbakhsh 2008. Modeling and Combining Access Control Policies Using Constrained Policy Graph, *Journal of Applied Sciences*, Vol. 8, and pp. 3561-3571. doi:10.3923/jas.2008.3561.3571
[9]. Object-Z/CSP Specification 2000. *Proceedings of 2nd Inter- national Conference on Integrated Formal Methods*, Springer Verlag, London, Vol. 1945, pp. 194-213.
[10]N. A. Zafar and F. Alsaade, 2012. Syntax-Tree Regular Expression Based DFA Formal Construction, *Intelligent Information Management* (*IIM*), Vol. 4, No.4 and pp. 138- 146.doi.org/10.4236/iim.2012.44021
[11] N. A. Zafar, A. Hussain and A. Ali, 2010. Verifying Monoid and Group Morphisms over Strongly Connected Algebraic Automata, *Journal of Software Engineering and Applications*, Vol. 3, No. 8 and pp. 803-812. doi.org/10.4236/jsea.2010.38093
[12] N. A. Zafar, N. Sabir and A. Ali, 2008. Construction of Intersection of Nondeterministic Finite Automata using Z Notation, *International Journal of Electrical and Computer Engineering*, Vol. 3, No. 2 and pp. 96-101.
[13] N. A. Zafar 2009. Formal Specification and Validation of Railway Network Components Using Z Notation, *IET*, *Software*, Vol. 3, No. 4 and pp. 312-320. doi.org/10.1049/iet-sen.2008.0082
[14] N. A. Zafar, A. Hussain and A. Ali 2009 Refinement: Formal Proof of Equivalence in Endomorphisms and Automorphisms over Strongly Connected Automata, *Journal of Software Engineering and Applications*, Vol. 2, No. 2, 2009, pp. 77-85. doi.org/10.4236/jsea.2009.22012
[15] S. Dupuy, Y. Ledru and M. Chabre-Peccoud 2000. An Overview of RoZ: a Tool for Integrating UML and Z Specifications, in 12th Conference on Advanced information Systems Engineering-CAiSE'2000, volume 1789 of Lecture Notes in Computer Science, Springer-Verlag, Stockholm.
[16] X. He 2000. Formalizing UML Class Diagrams: A Hierarchical Predicate Transition Net Approach, Procee- dings of 24th Annual International Computer Software and Applications Conference, Taipei, (25-28 October 2000), pp 217-222
[17] H. Leading, J. Souquieres, 2002. Integration of UML Views using B Notation, Workshop on Integration and Transformation of UML models (WITUML02).
[18]. Aftab Ali Haider and Aamer Nadeem, 2013. A Survey of Safety Analysis Techniques for Safety Critical Systems, International Journal of Future Computer and Communication, Vol. 2, No. 2, (April 2013), pp 134-137. DOI: 10.7763/IJFCC.2013.V2.137
[19]. K.Ranjini, A.Kanthimath, Y.Yasmine, 2011. Design of Adaptive Road Traffic Control System through Unified Modeling Language, in International Journal of Computer Applications (0975 – 8887) Volume 14– No.7 (February 2011), pp 36-41
[20] Dr. A.K.Sharma, Monika Singh, 2013 "Comparison of the Formal Specification Languages Based Upon Various Parameters", IOSR Journal of Computer Engineering (IOSR-JCE), Volume 11, Issue 5 (May. - Jun. 2013), PP 37-39. Doi: 10.9790/0661-1153739.
[21]. Emanuel S. Grant, Vanessa K. Jackson, and Sophine A. Clachar , 2012 "Towards a Formal Approach to Validating and Verifying Functional Design for Complex Safety Critical Systems" GSTF Journal on Computing (JoC) Vol.2 No.1(April 2012), pp 202-207, doi: DOI: 10.5176_2010-2283_2.1.151.
[22] I. Meisels, M. Saaltink, 1997 "The Z/EVES Reference Manual, TR-97-5493- 03," ORA Canada, CANADA.
[23]. E. A. Boiten, J. Derrick, G. Smith, 2004. Integrated Formal Methods (IFM 2004), Canterbury, UK, Springer.
[24] J. Davies, J. Gibbons, 2007. Integrated Formal Methods (IFM 2007), Oxford, UK, Springer-Verlag.
[25] J. Romijn, G. Smith, J. v. d. Pol ,2005. Integrated Formal Methods (IFM 2005), Netherlands, Springer.
[26] K. Araki, A. Galloway, K. Taguchi, 1999. Integrated Formal Methods (IFM 99), York, UK, Springer-Verlag.
[27] M. Butler, L. Petre, K. Sere, 2002.Integrated Formal Methods (IFM 2002), Turku, Finland, Springer-Verlag.
[28] W. Grieskamp, T. Santen, B. Stoddart, "Integrated Formal Methods (IFM 2000)," Germany, Springer-Verlag.
[29].J. M. Spivey, 1989 "The Z Notation: A Reference Manual," Prentice-Hall, Englewood Cliffs.
[30]. J. M. Spivey, 1988. Understanding Z: a Specification Language and its Formal Semantics, volume 3 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (January 1988).